Learning Cost Benefit Trade-offs

Tim Menzies[1]
University of British Columbia, Canada
tim@menzies.com

Martin S. Feather[2]
Jet Propulsion Laboratory, USA
Martin.S.Feather@Jpl.Nasa.Gov

March 5, 2002

1 Introduction

The more we know about software costs and benefits, the more precisely we can analyze the trade-offs between options in a software project. Sadly, in the usual case, precise knowledge is absent and analysts must make do with what little information is known. But how bad can that be? That is, how *little* information can we have and still be able to make some definite conclusions?

Our experience has been that, in early lifecycle, the *entire space* of options is known. What is unknown, however, is the domain knowledge that *restricts* that space. For example, while we might not predict exactly the lines of code in the system we are building, we can offer upper and lower bounds on that size.

For some years now, we have been exploring early lifecycle cost-benefit choices that surveys and summarizes this space of options. Our method uses the following techniques:

- *Disjunctive modelling*: In disjunctive modelling, when we don't know, we include the whole range as possible values for a variable.
- *Monte Carlo simulation*: Simulators are built which, when accessing a variable for the first time, selects and caches a random value from the possible range. If that variable is ever accessed twice, then the cache is used to return the same value as selected early in the simulation.

- *Machine learning to summaries the simulations*: The Monte Carlo simulations generate too much data for a human analyst to read and understand. Machine learners can automatically find the *smallest* number of variables that *most influence* the outcome of the model.

In the best case, the machine learner finds emergent stable conclusions from within the space of possible behaviors. In the worst case, no such stable conclusions exist and the learnt summaries will not be enlightening. This worst case result has yet to be seen, and we have theoretical reasons for believing that, on average, we should expect to find a small number of variables that control the larger space of all options [9, 12]. For some years now, we have repeatedly observed a curious *narrow funnel effect*. In many domains, it has been observed that a small number of critical variables control the remaining variables within a system, the metaphor being that all processing runs down the same narrow funnel [10]. The concept of narrow funnels has been reported in many domains under a variety of names including: *master-variables* in scheduling [1]; *prime-implicants* in fault-tree analysis [6]; *the dominance filtering* used in Pareto optimization of designs [5]; and the *base controversial assumptions* of HT4 [8]. Whatever the name, the core intuition in all these terms is the same: what happens in the total space of a system can be controlled by a small critical region. Where the narrow funnel effect exists, the space of options within a large space reduces to just the range of a few variables within the narrow funnel.

The rest of this paper offers case studies with this technique for assessing the cost and benefits of early life cycle software project options. Two machine learning summa-

---
[1]Lane Department of Computer Science, University of West Virginia, PO Box 6109, Morgantown, WV, 26506-6109, USA http://tim.menzies.com.

[2]Jet Propulsion Laboratory, USA California Institute of Technology, 4800 Oak Grove Dr, Pasadena CA 91109-8099

		current situation	proposed changes
prec = 0..5	precedentness	0, 1	
flex = 0..5	development flexibility	1, 2, 3, 4	1
resl = 0..5	risk resolution	0, 1, 2	2
team = 0..5	team cohesion	1, 2	2
pmat = 0..5	process maturity	0, 1, 2, 3	3
rely = 0..4	required reliability	4	
data = 1..4	database size	2	
cplx = 0..5	product complexity	4, 5	
ruse = 1..5	level of reuse	1, 2, 3	3
docu = 0..4	doco requirements	1, 2, 3	3
time = 2..5	runtime constraints	?	
stor = 2..5	main memory storage	2, 3, 4	2
pvol = 1..4	platform volatility	1	
acap = 0..4	analyst capability	1, 2	2
pcap = 0..4	programmer capability	2	
pcon = 0..4	programmer continuity	1, 2	2
aexp = 0..4	analyst experience	1, 2	
pexp = 0..4	platform experience	2	
ltex = 0..4	experience with tools	1, 2, 3	3
tool = 0..4	use of software tools	1, 2	
site = 0..5	multi-site development	2	
sced = 0..4	time before delivery	0, 1, 2	2
	# of combinations=	$6 * 10^6$	

Figure 1: A NASA software project. Unknowns in the current situation are shown as ranges or, in the case of total lack of knowledge, a "?".

rization methods will be shown: TARZAN, and its descendant, TAR2.

2 Case Study 1: COCOMO II

Menzies & Sinsel explored a space of 54 million options to find two key control variables [13]. In that application, a COCOMO-based tool [7] was used to evaluate the risk that a NASA software project would suffer from development overrun (that project is shown in Figure 1).

The tool used in that study required a guesstimate of the source lines of code (SLOC) in the system and certain internal tuning parameters which, ideally, are learnt from historical data. Lacking such data, Menzies & Sinsel used three guesses for SLOC and three sets of tunings which they took from the literature.

In that study, feuding stakeholders proposed 11 changes to a project. Some of the project features were unclear and, for those features, project managers could
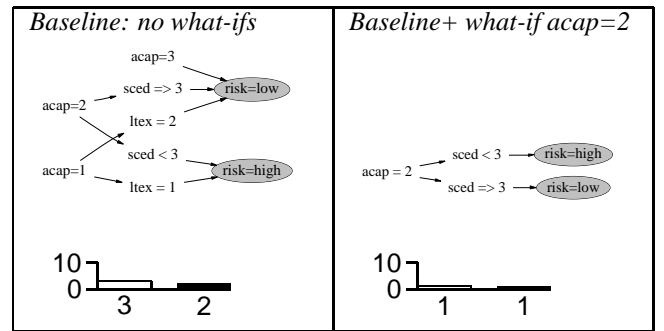

Figure 2: TOP: A decision tree (left) and a pruned tree (right) holding all branches that do not contradict *acap=2*. BOTTOM: Number of branches to different risk classification. Legend: ☐ =low risk ■ =high risk.

only offer ranges for the required inputs to the COCOMO-based tool. These ranges offered 2930 possible combinations for the inputs. When combined with the other uncertainties, this generated a space of 54 million possibilities ($2930 * 2^{11}$*three guesses for SLOC * three tunings).

Faced with this overdose of possibilities, Menzies & Sinsel performed 50,000 Monte Carlo simulations where the inputs were taken from the 54 million possibilities. A machine learning program generated decision trees from the 50,000 runs. A tree query language called TARZAN then swung through the learnt trees looking for the least number of attribute ranges that had the biggest impact on the overall software development risk.

TARZAN treated the learnt trees as a space of possibilities within the logged behavior. TARZAN ran what-if queries by pruning all branches in the learnt trees that contradicted some what-if possibility. For example, if we wonder "what-if acap=2", then Figure 2, top left, would be pruned to Figure 2, top right. This particular "what-if" turns out to be a bad idea. The histograms in Figure 2, bottom, show that this pruning drives us into a situation where the ratio to low risk to high risk projects changes from 3:2 to 1:1. That is, if *acap=2*, then we increase our chances of a high-risk project.

Figure 3 shows some of the what-if queries conducted over the trees learnt from the 50,000 runs. The baseline risk profile is shown in cell A1 of Figure 3: prior to the what-if queries, the learnt trees hold branches to 7,24,8
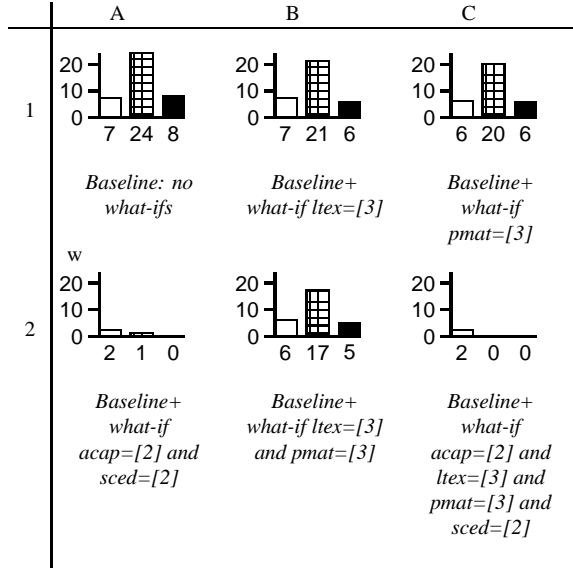
	A	B	C
1	20 10 0 7 24 8	20 10 0 7 21 6	20 10 0 6 20 6
	Baseline: no what-ifs	*Baseline+ what-if ltex=[3]*	*Baseline+ what-if pmat=[3]*
2	w 20 10 0 2 1 0	20 10 0 6 17 5	20 10 0 2 0 0
	Baseline+ what-if acap=[2] and sced=[2]	*Baseline+ what-if ltex=[3] and pmat=[3]*	*Baseline+ what-if acap=[2] and ltex=[3] and pmat=[3] and sced=[2]*

Figure 3: Number of branches to different risk classifications. Legend: ☐ =low risk ▦ =medium risk ■ =high risk.

low,medium,high risk projects respectively. Seven of the proposed changes had little impact on the baseline. Of the remaining four proposed changes, two are clearly superior. Cell A2 shows that that having moderately talented analysts and no schedule pressure (*acap=[2], sced=[2]*) reduced the risk in this project nearly as much as any other, larger subset. Exception: B2 applies actions to remove all branches to medium and high risk projects. Nevertheless, Menzies & Sinsel recommended A2, not B2, since A2 seemed to achieve most of what B2 can do, with much less effort.

Note that Figure 3 takes $\frac{1}{6}$th of a page to display and shows the key factors that control the classifications of 54,000,000 possibilities. This astonishing reduction in the argument space is consistent with the COCOMO-based tool containing narrow funnels.

3 Case Study 2: JPL models

Analysts at the NASA Jet Propulsion Laboratory sometimes debate satellite design by building a semantic network connecting design decisions to satellite requirements [2]. This network links *faults* and risk mitigation *actions* that effect a tree of *requirements* written by the stakeholders. Potential faults within a project are modelled as influences on the edges between requirements. Potential fixes are modelled as influences on the edges between faults and requirements edges.

This kind of requirements analysis seeks to *maximize* our coverage of the requirements while *maximizing* the ways the actions reduce the impact of the faults and *minimizing* the costs of the actions. Optimizing on all these criteria is complicated by the interactions inside the model. For example, in Figure 4, *fault2* and *require4* are inter-connected: if we cover *require4* then that makes *fault2* more likely which, in turn, makes *fault1* more likely which reduces the contribution of *require5* to *require3*.

The net can be executed by selecting actions and seeing what benefits results. One such network included 99 possible actions; i.e. $2^{99} \approx 10^{30}$ combinations of actions. Note the black line, top-left, of Figure 5. All the dots below this line were generated via 10,000 random selections of the decisions, and the collection of their associated costs and benefits. All the dots above this line represent high benefit, low cost projects found by the TAR2 machine learner [4] described in the appendix. In a result consistent with funnel theory, the learner could search a space of 10^{30} decisions to find 30 (out of 99) that crucially effected the cost/benefit of the satellite. Note that this means TAR2 also found 99-30=67 decisions that could be ignored.

For comparison purposes, a genetic algorithm (GA) was also applied to the Figure 5 domain [4]. The GA also found decisions that generated high benefit, low cost projects. However, each such GA solution commented on every possible decisions and there was no apparent way to ascertain which of these are the most critical decisions. The TAR2 solution was deemed superior to the GA solution by the domain experts, since the TAR2 solution required just 30 actions.

4 Conclusion

Even when faced with incomplete information, it may still be possible to find stable conclusions about cost-benefit trade-offs.
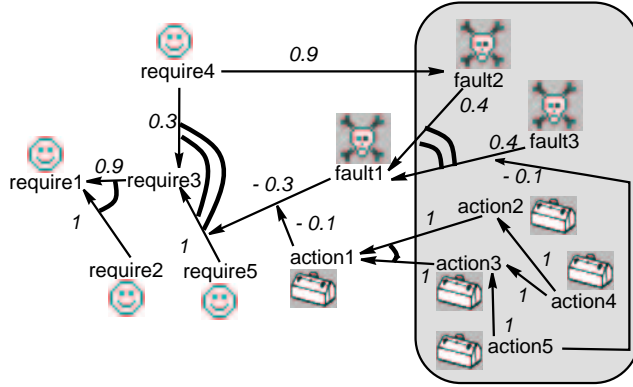
Faces denote requirements;

Toolboxes denote actions;

Skulls denote faults;

Conjunctions are marked with one arc; e.g. *require1* if *require2* and *require2*.

Disjunctions are marked with two arcs; e.g. *fault1* if *fault2* or *fault3*.

Numbers denote impacts; e.g. *action5* reduces the contribution of *fault3* to *fault1*, *fault1* reduces the impact of *require5*, and *action1* reduces the negative impact of *fault1*.

Oval denotes structures that are expressible in the latest version of DDP (under construction).

Figure 4: Left: an DDP-style [3] software management oracle. Right: explanation of symbols.

APPENDIX: The TAR2 machine learner

Classical machine learning (e.g. C4.5 [14]) can be applied to learn implications between attribute ranges and results (e.g.):

$$X > 1 \land Y < 0 \to class = highCostProject$$

However, if applied to a non-trivial requirements interaction model a large number of such implications result. Some form of summarization is required.

One way to do this is to study pairs of rules that lead to different results and reporting the changes to attribute ranges that change (e.g.) a $highCostProject$ into a $lowCostProject$. TARZAN implemented such a search as a post-processor to C4.5. TAR2 performs the same search directly, without needing C4.5 [11]. Starting with examples, TAR2 finds range settings that are highly associated with some "good" outcome (e.g $lowCostProject$) and not highly associated with some "bad" outcome (e.g. $highCostProject$).

TAR2 outputs implications of the form (e.g.)

$$X > 1 \land Y < 0 \to less \text{ "bad" and more "good"}$$

where "less" and "more" are measures of the change in the frequency of "good" and "bad" before and after applying $X > 1$ and $Y < 0$ to the examples. The set of
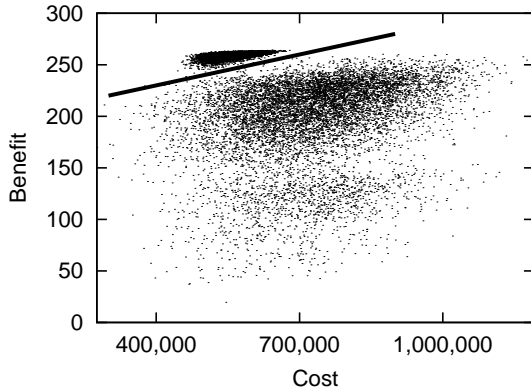

Figure 5: Results from the satellite domain. The dots below the line show the initial output of the model: note the very large spread in the costs and benefits. The dots above the line show the final outputs of the model after 5 iterations of TAR2 learning.

attribute ranges ($X > 1$ and $Y < 0$) is called a *treatment*. Such treatments are the constraints that TAR2 is proposing on future actions in order to increase the chances of less "bad" and more "good".

References

[1] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

[2] M. Feather, S. Cornford, and T. Larson. Combining the best attributes of qualitative and quantitative risk management tool support. In *15th IEEE International Conference on Automated Software Engineering, Grenoble, France*, pages 309–312, September 2000.

[3] M. Feather, H. In, J. Kiper, J. Kurtz, and T. Menzies. First contract: Better, earlier decisions for software projects. In *Submitted to the ACM CIKM 2001: the Tenth International Conference on Information and Knowledge Management*, 2001. Available from `http://tim.menzies.com/pdf/01first.pdf`.

[4] M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *RE'03 (submitted)*, 2002.

[5] J. Josephson, B. Chandrasekaran, M. Carroll, N. Iyer, B. Wasacz, and G. Rizzoni. Exploration of large design spaces: an architecture and preliminary results. In *AAAI '98*, 1998. Available from `http://www.cis.ohio-state.edu/~jj/Explore.ps`.

[6] R. Lutz and R. Woodhouse. Bi-directional analysis for certification of safety-critical software. In *1st International Software Assurance Certification Conference (ISACC'99)*, 1999. Available from `http://www.cs.iastate.edu/~rlutz/publications/isacc99.ps`.

[7] R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.

[8] T. Menzies and P. Compton. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175, 1997. Available from `http://tim.menzies.com/pdf/96aim.pdf`.

[9] T. Menzies and B. Cukic. Adequacy of limited testing for knowledge based systems. *International Journal on Artificial Intelligence Tools (IJAIT)*, June 2000. Available from `http://tim.menzies.com/pdf/00ijait.pdf`.

[10] T. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*, 1999. Available from `http://tim.menzies.com/pdf/99re.pdf`.

[11] T. Menzies and Y. Hu. Reusing models for requirements engineering. In *First International Workshop on Model-based Requirements Engineering*, 2001. Available from `http://tim.menzies.com/pdf/01reusere.pdf`.

[12] T. Menzies and Y. Hu. Just enough learning (of association rules). In *KDD'02 (submitted)*, 2002. Available from `http://tim.menzies.com/pdf/02tar2.pdf`.

[13] T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00ase.pdf`.

[14] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

5